APPLICATION FOR U.S. LETTERS PATENT

TITLE:

STREAM-DOWN PREFETCHING CACHE

INVENTORS:

HERBERT HING-JING HUM
ZOHAR BOGIN

PREPARED BY:

KENYON & KENYON
333 W. SAN CARLOS STREET
SUITE 600
SAN JOSE, CA 95110

(408) 975-7500

Docket No. 2207/783002

STREAM-DOWN PREFETCHING CACHE

Related Application

This patent application is a continuation application of, and claims priority to, U.S. Application Serial No. 09/541,392, filed March 31, 2000.

Field of the Invention

The present invention relates to computer memory. More particularly, the present invention relates to stream-down prefetching data into a cache memory.

Background of the Invention

Most modern computer system architectures include a cache. That is, a level of the memory hierarchy between the processor and the main memory. Generally, access to data stored in a cache is much faster than the time needed to retrieve data from main memory. However, cache memory is typically more expensive and smaller than main memory. A common cache analogy is that main memory is like books on shelves in a library and a cache is like a small subset of those books on a desk. The widespread use of cache memory is a clear statement that it does increase system performance, more than enough to justify added complexity.

It is possible to have multiple levels of cache, or more correctly multiple levels of memory, in a memory hierarchy. Generally, in such a system, all the data is stored at the lowest level, which is typically the largest in size and the slowest to access. Data is then copied to the higher levels, which may decrease in size and increase in speed.

Designing a memory hierarchy for a general purpose computer, including algorithms for what data should be stored in cache is a complex process that has received a lot of attention over the years. Empirically, computer programs have a tendency to reuse data that has been accessed recently (temporal locality) and also to access data located near data that has been accessed recently (spatial locality). Many memory hierarchies utilize the existence of these localities for storing data in a cache. For example, keeping recently accessed data in a cache exploits temporal locality. Retrieving blocks of data, instead of an individual word, into a cache exploits spatial locality. This disclosure will use the term "data" to include both the traditional concept of data, such as alpha-numeric constants or variables, set or read by a computer program, as well as program instructions. For the memory hierarchy, both types of data are merely values stored at a particular location and the ultimate uses of the individual stored values are irrelevant. Some architectures do, however, maintain separate caches for traditional data and instructions, and must distinguish between them.

Ideally, whenever a processor calls for data from memory, that data will be found in the cache. However, cache misses do occur. Two common causes of cache misses are cold-start misses and capacity misses. A cold-start miss occurs when a processor first accesses a particular range of data addresses. A capacity miss occurs due to limited cache sizes, such as when previously cached data is "bumped" from the cache by newer data just before it is needed. A larger cache may decrease the capacity miss rate, but have no effect on the cold-start miss rate.

Brief Description of the Drawings

Figure 1 is a diagram of a prefetch cache in accordance with an embodiment of the present invention.

Figure 2 is a diagram of a request history buffer in accordance with an embodiment of the present invention.

Figure 3A-3C depict a stream-up data access trend in accordance with an embodiment of the present invention.

Figure 4A-4D depict an out-of-order stream-up data access trend in accordance with an embodiment of the present invention.

Figure 5A-5C depict a stream-down data access trend in accordance with an embodiment of the present invention.

Figures 6A-6C depict an out-of-order stream-down data access trend in accordance with an embodiment of the present invention.

Figure 7 is a block diagram of a computer system in accordance with an embodiment of the present invention.

Detailed Description

An embodiment of the present invention is directed to prefetching data into a cache, based on a history of data requests. The data prefetching is based on a trend, upward or downward, in the memory addresses of previously requested data. This prefetching is referred to as stream-up or stream-down, depending on the trend direction. An embodiment of the present invention uses a request history buffer (RHB) to store and detect trends in data requests. A prefetch cache is preferably used to store the prefetched data. The prefetch cache may be adapted for use in conjunction with other cache memories in the system.

Many memory hierarchies save recently requested data in a cache in order to take advantage of temporal locality in data requests. Also, many memory hierarchies retrieve blocks of contiguous data to take advantage of the spatial locality in data requests. However, neither of these techniques explicitly attempts to dynamically anticipate a data request trend and prefetch data based on such a trend.

An embodiment of the present invention prefetches data, whether or not a data request trend has been established, defaulting to stream-up prefetching. That is, the next higher memory address is prefetched, if not already in cache, with each data request, unless a downward trend has been established. If there is a downward trend in the memory address requests, the data from the next lower address is prefetched.

Memory hierarchy systems that retrieve an entire block of data, in order to take advantage of spatial locality, may achieve the result of the present invention in some cases, but only because such a brute force retrieval blindly brings a large amount of data in a cache. The present invention, by being much more selective in prefetching data, does not bring in as much extraneous data. Also, block data retrieval will often result in a cache miss when the next data address is in the adjacent block of data. Prefetching may be more efficient than block data retrieval, and more accurately anticipate spatial locality for requested data at the border of a memory block. Also, prefetching data anticipates a data retrieval request. Retrieving an entire block of data ignores trends in the direction of data request addresses. As such, the present invention uses more information to better anticipate future data requests.

The present invention also lowers the cold-start miss rate by examining the history of data requests, anticipating what data that will be requested and prefetching that data. That is, data is retrieved prior to the data being explicitly called by a processor, and decreasing the overall number of cache misses. The prefetching algorithm in accordance with an embodiment of the present inventor may be biased toward data with numerically higher addresses (stream-up prefetching) in memory. However, stream-down prefetching may be used when such a pattern of data requests is found in the data request history.

Sectored caches are cache architectures that store data from multiple sectors of contiguous memory locations within a single line of cache. For example, a two-sectored cache with a 32 bytes cache line would store data from two contiguous memory sectors of 16 bytes each within a single line of cache. A more general description of such sectored caches would be an N-sectored cache, where N could be any positive integer value, storing N sectors per line of cache. Typically, N will equal a small integer power of two. However, the present invention is not intended to be limited to any particular value of N. As will be shown below, values of N larger than one complicate the prefetching slightly and must be explicitly considered. The examples described below use a two-sectored cache. Those of ordinary skill in the art, with the benefit of this disclosure, will understand how embodiments of the present invention can be easily adapted for other values of N. An additional complication to the prefetching is the processing of instructions out-of-order. A technique used in some modern computer architectures is to look ahead and speculatively execute instructions before the instructions would occur in the program sequence. These instructions are speculatively executed in the sense that conditional branching may occur and the out-of-order instruction may not be needed. However, the processing of these out-of-order instructions may affect both the data in the system cache and the RHB, making it more difficult to identify trends. An embodiment of the present invention may be used with a system architecture that processes instructions out-of-order, as well as architectures that do not. The preferred cache size may be adjusted based on how far a particular system looks ahead to process out-of-order instructions.

Figure 1 describes a prefetch cache 2 in accordance with an embodiment of the present invention. Prefetch cache 2 preferably uses content addressable memory (CAM), which may also be called associative memory. A CAM requires that each entry of data field 4 be mapped to its corresponding memory location by a tag 6, preferably the same as the memory location address used by the processor to retrieve data from memory. A CAM type cache organization is known to those of ordinary skill in the art.

The RHB, unlike prefetch cache 2, does not store data from memory. Rather, RHB flags whether or not data from a particular sector has been retrieved. The format of the RHB follows from that of the prefetch cache 2. A RHB for use with an N-sector prefetch cache will typically have N flags, or data fields, in addition to an address tag field. The flags in the RHB may be a single bit. **Figure 2** describes a RHB 8 in accordance with an embodiment of the present invention. RHB 8 in **Figure 2** is designed to work with a two-sectored cache, although the present invention is not intended to be limited to any particular number of sectors or cache organization. The RHB has two data fields, 10 and 12, per line. Tag 14 in RHB 8 may be one bit shorter than tag 6 in prefetch cache 2. Preferably, the low bit of the data memory location address will map directly to either data field 10 or 12.

The present disclosure will largely ignore cache management details such as validity bits, selecting optimal cache sizes, and writing cache data back to memory, as such issues are known to those of ordinary skill in the art and the present invention is not intended to be limited to any particular cache organization.

Figures 3A-3C illustrate an embodiment of the present invention performing stream-up prefetching with a two-sector cache. RHB 8 is a buffer for storing the history of data requests. Initially, data is called from a sector of memory identified by X (X is a memory address modulo the sector size), and this memory address, X, is stored in RHB 8. A bit is set in data field 10 to indicate address X has been called. Next, RHB 8 is checked for a tag to indicate the presence of data from sector X + 2 and prefetch cache 2 is checked for the data. For example, if a sector is 16 bytes and X is 120 hexadecimal, then X + 2 is 140 hexadecimal. Because the bit for sector X + 2 is not indicated in RHB 8, nor is the data in prefetch cache 2, the stream-up prefetching direction is chosen as the default direction. With a two-sector cache, the sector for prefetching is two sectors away from the sector containing the data address explicitly called by the processor. If the data from sector

X + 2 is not indicated in either RHB 8 or prefetch cache 2, it is retrieved from memory, or prefetched, and stored in prefetch cache 2.

Figure 3B depicts contents of RHB 8 and prefetch cache 2 following the request of data from sector X + 1. There is a trend of requesting data from increasing memory addresses, so there is no need to change from the default direction of stream-up prefetching. RHB 8 and prefetch cache 2 are then checked for the presence of sector X + 3, the next prefetch sector for stream-up prefetching in a two-sectored cache. If X + 3 is not found, it is prefetched and stored in prefetch cache 2. A bit is set in data field 12 to indicate the presence of data from section X + 1.

In **Figure 3C**, there has been a request for data from memory address X + 2. This time, the requested data is present in prefetch cache 2 and does not have to be retrieved from memory. The requested data is dispatched from prefetch cache 2, and sector X + 4 is prefetched, if it is not already present in prefetch cache 2 or indicated in RHB 8.

Note that RHB 8 which is to be used with a two-sectored cache, has two (single bit) data entries per line in **Figures 3A-3C**. In an embodiment of the present invention with a four-sectored cache, RHB 8 would have four (single bit) data entries per line. Note also that there is a preferred, and consistent, location for each data entry with RHB 8. That is, the upper portion of RHB 8 is used to indicate the presence of even sector addresses (such as X) and the lower portion is used for the odd addresses (such as X + 1). The present invention is not intended to be limited to any particular address allocation scheme within RHB 8, only that the scheme chosen remain consistent.

The notation used in **Figures 3A-3C**, as well as those that follow, is a simplified form of the actual entries in prefetch cache 2. Only tag 6 field of prefetch 2 is used in the examples to signify that data in a particular sector has been prefetched. The actual data, stored in data 4 field, has been omitted for clarity.

The device that prefetches data from memory may be called the "prefetcher". The device that works in coordination with the prefetcher to set bits in data fields 10 and 12 of RHB 8 may be called the "prefetch control unit".

Figures 4A-4C also illustrates an embodiment of the present invention performing stream-up prefetching. In **Figure 4A**, data is initially requested from sector X + 1. This information (request/presence of sector X + 1) will be stored in the lower portion of RHB 8, in contrast to the upper portion used to indicate the presence of sector X in **Figure 3A**. Stream-up prefetching is again selected as the default prefetch direction when there is insufficient information to establish a trend. RHB 8 and prefetch cache 2 are checked for the presence of data from sector X + 3 in either. If it is not, sector X + 3 is prefetched and stored in prefetch cache 2.

Next, as shown in **Figure 4B**, data is required from sector X, as indicated by the bit set in data field 10 of RHB 8. Although this data request is to a lower memory address than the initial request for X + 1, an embodiment of the present invention continues stream-up prefetching, searching for sector X + 2. Since the sector X + 2 is not already present in either RHB 8 or prefetch cache 2, it is retrieved from memory and stored in prefetch cache 2.

Figure 4C shows the next request for data from sector X + 2. This data is then dispatched from prefetch cache 2, and prefetching continues on sector X + 4, if it is not already indicated or present in either RHB 8 or prefetch cache 2.

The process illustrated in **Figures 4A-4C** may occur with computer architectures using out-of-order instruction processing. That is, although the data requested does not initially appear to be consistent with a stream-up trend, this could be the result of an out-of-order instruction requesting data.

Figures 5A-5D illustrate stream-down prefetching in accordance with an embodiment of the present invention. **Figures 5A** and **5B** are identical to **Figures 4A** and **4B**. The process explicitly requests data from sectors X + 1 and X, with data from sectors X + 3 and X + 2 then prefetched. In **Figure 5C**, however, the stream-down trend continues with a request for data from sector X - 1, which is retrieved from memory, and the request noted in RHB 8. The request for data from sector X - 1 triggers a check for the presence of sector X + 1 in both prefetch cache 2 and RHB 8. The bit set in data field 12 of RHB 8 indicates sector X + 1 has been called. The presence of X + 1 in RHB 8 is used as an indication of a trend towards lower memory addresses, or a call for stream-down prefetching. Next, using a stream-down prefetch direction, the sector X - 3 is prefetched, and stored in prefetch cache 2, if its presence is not flagged in RHB 8 or it is not already present in prefetch cache 2.

In **Figure 5D** data is called from sector X - 2, continuing in the stream-down direction. In response to the request for X - 2, RHB 8 and prefetch cache 2 are checked for the presence of sector X. Since it is present in RHB, sector X - 4 is prefetched and stored in prefetch cache 2, if either the flag is not already present in RHB 8 or the data is in prefetch cache 2.

Figures 6A-6C also illustrates an embodiment of the present invention performing stream-down prefetching. Initially there is a request for data from sector X. The presence of sector X is indicated in RHB 8. The default stream-up prefetching, with a two-sector cache, looks for the presence of sector X + 2 in RHB 8 or the data in prefetch cache 2. Since sector X + 2 is not present in either RHB 8 or prefetch cache 2, it is prefetched and stored in prefetch cache 2. Next, the processor calls for data from sector X + 1, which also will be retrieved from memory and its presence noted in RHB 8. In response to the request for data in sector X + 1, data from sector X + 3 is prefetched, since it is not already present in prefetch cache 2 or its presence indicated in RHB 8. In **Figure 6C** there is a request for data from sector X - 1, which will be retrieved from memory and its presence indicated in RHB 8. The data from sector X + 1 would be prefetched, in a (default) stream-up prefetch using a two-sector cache, in response to a request for data from sector X - 1 by the processor. However, a flag for sector X + 1 is already present in RHB 8. The indication of sector X + 1, a numerically higher memory address, is used by an embodiment of the present invention to switch to stream-down prefetching. In this stream-down prefetching mode, data from sector X - 3 is prefetched and stored in prefetch cache 2.

In the process illustrated in **Figures 6A-6C**, there are calls for data from sectors X, X + 1 and then X - 1. The overall trend towards lower memory location addresses, is obscured somewhat by the second data request. However, out-of-order instruction processing can produce just such an effect. An embodiment of the present invention examines RHB 8 and prefetch cache 4 to determine the overall trend. That is, whether to use the default stream-up direction or to switch to stream-down prefetching.

An embodiment of the present invention uses larger prefetch cache 2 size (32 cache lines) than the associated RHB 8 size (8 or 16 entries) where the prefetch cache 2 is fully-associative and employs a least-recently used replacement policy. Such a differential sizing may eliminate the need to coordinate cache replacement between prefetch cache 2 and RHB 8.

Figure 7 show a system architecture with two levels of cache (16 and 18) between processor core 20 and main memory 22, in addition to prefetch cache 2 and the associated RHB 8. The examples illustrated in this description are for the case where the level 2 cache 18 is a two-sector cache. That is, the prefetch cache 2 and RHB 8 as described can capture streaming patterns from a two-sector cache. Other embodiments of the present invention may use a prefetch cache 2 between the two levels of cache (16 and 18), either in addition to or instead of, prefetch cache 2 shown in **Figure 7**. Similarly, the present invention is not intended to be limited to any particular number of cache levels.

The present invention may be implemented in software or firmware, as well as in programmable gate array devices, ASICs and other hardware.

Although various embodiments are specifically illustrated and described herein, it will be appreciated that modifications and variations of the present invention are covered by the above teachings and within the purview of the appended claims without departing from the spirit and intended scope of the invention. Moreover, the present invention applies to a broad range of computer architectures, and is therefore a general approach that includes a broad range of specific implementations.